REMARKS

This Amendment is responsive to the Office Action mailed from the U.S. Patent and Trademark Office on September 13, 2004, (hereinafter, "the Office Action"). The Office Action rejected all then-pending claims under 35 U.S.C. § 103(a).

In response to the Office Action, claims 1-4, 10-13, and 16-18 are amended. New claims 21 and 22 are added. Upon entry of this Amendment, claims 1-22 will be pending. No new matter is added by way of the amended claims, which are fully supported by the specification and drawings. Specifically, the Amendments to the independent claims moved limitations directed toward the loop unrolling aspect of the invention to dependent claims and clarifies certain details of the dynamic compiler as shown and described in the written description with respect to Figure 1.

Claim Rejections Under 35 U.S.C. § 103(a)

Claims 1-3, 7-13, and 16-20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent 6,539,541 to Geva (Geva) in view of U.S. Patent 6,170,083 to Adl-Tabatabai (Adl-Tabatabai). Applicants respectfully traverse because the prior art references do not teach or suggest each of the elements set forth in the claims, and because the prior art lacks motivation to combine or modify the references.

Claim 1 sets forth, inter alia, a method for executing a computer program using a dynamic compiler system wherein a byte code is received, a determination is made whether native code is available corresponding to the byte code, and when native code is not available, a counter is incremented. The counter tracks the number of times the byte code is interpreted, and when it is less than a threshold, the byte

code is interpreted. When it is greater than a threshold, then the routine including the byte code is compiled. Independent claim 10 sets forth a system having features similar to those set forth in claim 1 and independent claim 16 sets forth computer code performing steps similar to those set forth in claim 1.

Geva discloses a compiler for compiling source code into optimized machine-readable code. Geva does not suggest a dynamic compiler that determines whether or not to compile a specific code segment based on the number of executions of that code segment. The Office Action admits, "Geva doesn't expressly disclose executing interpreted byte code or counting the number of times each interpreted byte code is executed" (Office Action, page 4, lines 6-7).

Adl-Tabatabai does not overcome the deficiencies of Geva. Adl-Tabatabai teaches a dynamic compiler that optimizes performance based on path profiling. While Adl-Tabatabai does in fact count the frequency of execution of code segments, it does so only of *already compiled code segments*, (col. 6, lines 10-15) not of interpreted byte codes. Thus, the prior art references, including Adl-Tabatabai, fail to teach or suggest determining whether to compile a routine in byte code based on the number of executions of that routine. Instead, Adl-Tabatabai counts executions of a compiled code segment to determine the likely path of execution, i.e., jumps within the program, so that "hot" execution paths – paths executed most often, are dynamically optimized. Note that, for optimization, the optimized "hot path" is executed instead of the *original compiled object code* for improved performance. See the Abstract of Adl-Tabatabai, specifically the last 9 lines thereof.

Furthermore, while it is noted that Adl-Tabatabai does in fact disclose a Java interpreter and compiler, Adl-Tabatabai does not mention or suggest interpreting byte code segments and then compiling them based on frequency of execution. In fact, Adl-Tabatabai teaches away from interpreting byte codes altogether. Specifically, Adl-Tabatabai states, "Since the Java interpreter 331 interprets Java byte codes that are not native processor code, such Java interpreting tends to be slow. To improve performance of Java programs, Java "Just-In-Time" compilation was created." See column 3, lines 24-27.

Even if the combination of Geva and Adl-Tabatabai did disclose the features now set forth in the independent claims, there was no suggestion in the prior art to combine and/or modify the references as proposed in the Office Action. The Office Action states only that "it would have been obvious . . . to combine Geva and Adl-Tabatabai because implementing the system using byte codes would make the system platform independent" (page 4, lines 14-16 of the Office Action). The Office Action does not mention why it would be obvious to interpret the byte codes or compile based on the frequency of execution of interpreted byte codes. Furthermore, while Adl-Tabatabai suggests advantages of portability made possible through the use of the Java programming language, Adl-Tabatabai does not suggest counting the number of times code segments are interpreted for the purpose of compiling.

Since none of the references cited or of record in the present matter teach or suggest the elements set forth in the independent claims, Applicants respectfully submit that independent claims 1, 10, and 16, and all claims depending therefrom set forth subject matter not taught or suggested by the prior art. Therefore, Applicants submit that the present application is now in condition for allowance.

Claims 4-6, 14, and 15 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Geva in view of Adl-Tabatabai and further in view of U.S. Patent 5,457,799 to Srivastava (Srivastava). Applicants respectfully traverse.

Claims 4-6 depend from claim 1 and should therefore be allowed for the reasons discussed above with respect to claim 1. Likewise, claims 14 and 15 depend from claim 10 and should be allowed for the same reasons as claim 10 mentioned above.

However, even if Geva and Adl-Tabatabai were properly combined and met all the claim elements of the independent claims, claims 4-6 set forth additional features not taught or suggested by the prior art.

Claims 4-6 are directed to a tree data structure for storing information about various program segments in the computer program. Specifically, claim 4 sets forth, "building a loop tree based on loops included in the computer program" (lines 2-3). Claim 5 sets forth, inter alia, "wherein nested loops are represented in the loop as

child nodes." Claim 6 further limits claim 5, "wherein parallel loops are represented in the loop tree as nodes on the same level of the loop tree." Claims 13-15 are similar to claims 4-6, but are directed to a dynamic compiling system as opposed to a method per se.

Srivastava teaches employing a tree-like structure to represent code segments in a computer program (Figure 3). However, Srivastava does not teach a tree data structure, but a call graph. Trees are distinguished from call graphs in that edges of call graphs may arbitrarily directed. For example, nodes 324 and 322 in Figure 3 of Srivastava call each other and therefore have edges extending in opposite directions. This would be impossible in the tree structure where each child node is a loop nested within the parent node. Furthermore, Srivastava states specifically that, "very few real programs have call graphs in the form of a tree" (column 4, lines 15-16). Thus, the nodes of Strivastava do not represent loops, but procedures, and the edges of the call graph represent interfaces between the procedures. The interfaces may be a jump statement or other similar statement.

Since Srivastava is analyzing compiled native code, and not byte code, it must infer the presence of loops from the likelihood of repetition of a particular procedure call. See column 3, lines 53-57. Thus, not every node of Srivastava is a loop. The entire structure is distinct from the claimed loop tree.

Srivastava therefore teaches an entirely different data structure, which operates in a different way from that which is clearly set forth in the pending claims. Applicants respectfully submit that claims 4-6 and 14-16 are therefore allowable because the prior art references fail to teach or suggest the claim elements.

Accordingly, Applicants respectfully request the withdrawal of the 35 U.S.C. § 103(a) rejections and further submit that the references singly or in combination do not render the claimed invention obvious. Applicants respectfully request a Notice of Allowance based on the foregoing remarks. If the Examiner has any questions concerning the present amendment, the Examiner is kindly requested to contact the undersigned at (408) 774-6933. If any other fees are due in connection with filing this

amendment, the Commissioner is also authorized to charge Deposit Account No. 50-0805 (Order No. SUNMP017). A copy of the transmittal is enclosed for this purpose.

Respectfully submitted,
MARTINE & PENILLA, LLP

Leonard Heyman
Reg. No. 40,418

Martine & Penilla, LLP
710 Lakeway Drive, Suite 200
Sunnyvale, California 94086
Tel: (408) 749-6900

Customer Number 32291